Membros: Francisca Ariane dos Santos da Silva(554930) e Raíssa Ívyna Moreira de Oliveira(553679)

INTRODUÇÃO

O projeto baseia-se em um sistema de informação de linhas de ônibus com suas respectivas rotas. Foram implementadas funções em classes para fornecer informações aos passageiros que desejam saber se existem linhas de ônibus que passam por um determinado terminal ou partem dele com destino a uma cidade especificada pelo usuário. Com base nisso, os usuários inserem o destino desejado e o horário aproximado de chegada, e o sistema fornece informações sobre as linhas de ônibus, os quais compõem os atributos, como o número da linha, a companhia e os horários de partida e chegada.

A solução para o problema proposto, que é a necessidade de fornecer informações precisas e atualizadas sobre as linhas de ônibus disponíveis para os passageiros em um determinado terminal de ônibus,é a criação de um sistema de informações que armazena dados sobre as linhas de ônibus, assim como informações sobre as paradas ao longo das rotas das linhas de ônibus. Para implementar esse sistema, serão utilizadas estruturas de dados, serão a lista simplesmente encadeada e a lista duplamente encadeadas circulares. Os passageiros irão inserir o destino e o horário desejado, e o sistema fornecerá resultados com base nas funções usadas.

EXPLICAÇÃO DAS ESTRUTURAS DE DADOS

A estrutura de dados de lista simples é implementada no código fornecido como LinhaList. Esta estrutura consiste em nós de NodeLinhaList, onde cada nó representa uma Companhia e aponta para o próximo nó na lista. A primeira parte do código define a classe NodeLinhaList, que possui um atributo Companhia e um ponteiro next para o próximo nó. A LinhaList mantém um ponteiro para o primeiro nó firstNode na lista, o tamanho da lista size e um ponteiro para uma lista de paradas de ônibus associadas a essa linha, parada. Logo, a lista encadeada simples é usada para organizar as diferentes linhas de ônibus, e a lista de paradas está pontualmente associada a cada linha de ônibus.

Já a estrutura de dados de lista circular é implementada no código como hourRoundList. Nesta estrutura, cada nó (hourRoundNode) representa um horário de chegada e saída, bem como um nome associado, ou seja, cada nó contém informações sobre uma parada de ônibus. Como dito anteriormente a lista é circular, o que significa que o último nó aponta de volta para o primeiro, formando um ciclo e mantendo a sua circularidade. Cada nó tem um ponteiro next para o próximo nó e

um ponteiro prev para o nó anterior .A hourRoundList mantém um ponteiro para o primeiro nó firstNode um ponteiro para o último nó lastNode na lista e também mantém o tamanho da lista size.

EXPLICAÇÃO DA IMPLEMENTAÇÃO

As estruturas de dados definidas no código têm finalidades específicas, tais como A lista simples LinhaList serve para armazenar informações sobre diferentes companhias. Essa estrutura será proveitosa no projeto, pois manterá um registro de várias companhias que operam no sistema de transporte. Enquanto isso, a Estrutura hourRoundList é adequada para criar um ciclo de horários de chegada e saída, com os respectivos nomes associados. É significamente útil no agendamento da programação de horários que se repete ao longo do uso do menu. As estruturas de dados foram utilizadas para organizar e armazenar essas informações, permitindo operações como adicionar, remover e acessar paradas em linhas de ônibus.

Operações-base do projeto:

➢ A função inserirNaListaSimples é capaz de inserir uma nova companhia no final da lista. Ela cria um novo nó NodeLinhaList com o nome da companhia e o adiciona à lista.

➢ A função apagarNoIndexListaSimples atua na remoção de um nó de índice específico, ajustando os ponteiros next para manter a integridade da lista e o seu uso em outras funções contribui para a otimização do tempo no decorrer do código.

➢ A função removeParada é usada para remover uma parada de uma linha de ônibus específica. Ela começa encontrando a linha desejada usando pegarDoIndexListaSimples e em seguida, verifica se essa linha tem uma lista deparadas associadas. Se a linha não tiver paradas, uma mensagem informando que a linha não possui paradas é exibida.Se não, o algoritmo percorre a lista de paradas e remove a parada com o nome especificado pelo usuário. Se a parada for a primeira da lista, os ponteiros da lista de paradas são ajustados para remover a referência à primeira parada. Se a parada estiver em outra posição, a função percorre a lista de paradas e remove a parada desejada, ajustando os ponteiros apropriados.

➢ A função pegarDoIndexListaSimples permite acess
ar um nó da lista com base em seu índice. Isso facilita a recuperação de informações sobre uma companhia específica na lista.

➢ A função atribuirParadaEmLinha atribui uma nova parada a uma linha de ônibus específica. Ela começa criando um novo nó de parada e preenchendo suas informações. Em seguida, encontra a linha desejada usand pegarDoIndexListaSimples. Se a linha não tiver uma lista de paradas associada (parada), uma nova lista de paradas é criada. O novo nó de parada é então inserido

na lista de paradas da linha usando a função inserirNaListaCircular. Isso permite que novas paradas sejam atribuídas a uma linha de ônibus existente.
➢ A função inserirNaListaCircular terá a função de inserir um novo horário com nome associado no final da lista circular. Ela atualiza os ponteiros next e prev para manter a circularidade da lista.Tanto as estruturas de dados criadas como as operações definida são importantes para o armazenamento e manipulação eficiente de dados e informações no projeto, os quais foramusados para a listagem de companhias e programação de horários
➢ A função ImprimirLinhascomParada será essencial já que percorrerá a lista de linhas de ônibus, verificando se cada linha tem uma lista de paradas. Para cada linha com uma lista de paradas, ele imprime o nome da linha e, em seguida, chama a função ImprimirListaCircular para imprimir a lista de paradas associada a essa linha. Se a linha não tiver uma lista de paradas, uma mensagem informando que a linha não possui paradas é exibida para o maior esclarecimento possível do usuário.
A função alterarParada está correlacionada a uma função básica de acesso e essa função recebe como entrada uma lista de linhas de ônibus, um índice de linha, o nome atual de uma parada e o novo nome que se deseja atribuir à parada. Ela realizaa seguinte sequência de ações: primeiro, verifica se o índice da linha é válido e se a linha existe; em seguida, verifica se a linha possui uma lista de paradas associada; se essas condições são atendidas, a função itera pelas paradas da linha em busca da parada com o nome especificado. Quando a parada é encontrada, o nome é alterado para o novo nome fornecido. Em caso de sucesso, a função exibe uma mensagem informando que a operação foi bem-sucedida. Se a parada com o nome especificado não for encontrada, a função exibe uma mensagem de erro indicando que a parada não foi localizada na lista de paradas da linha.

COMPLEXIDADE

```
//Funcao que cria uma lista simples de Linhas de Onibus
// Ela aloca memoria para uma instancia de lineList, inicializa
// e retorna um ponteiro para a lista que foi criada
/* Complexidade O(1), pois possui operacoes fixas, com alocacao de
memoria e inicializacao da variaveis. O(1) + O(1) + O(1) + O(1) = O(4)
O(4) equivale a constante, portanto O(4) = O(1).
*/
lineList* criarListaSimples(){
        lineList* listHead = new lineList; //O(1)
        listHead->firstNode = NULL; //O(1)
        listHead->size = 0; // O(1)
        return listHead; // O(1)
}

//Funcao que permite adicionar uma nova Linha de Onibus
```

```
// a lista de Linhas criada criando um novo no de linha
// e inserindo no final da lista
/*
No pior caso, a complexidade dessa funcao eh O(n) pois ela verifica se a lista
esta vazia e assim faz O(1) em operacao, ja se nao estiver vazia, a funcao percorre
para encontrar o ultimo no da lista (O(n)), sendo n o numero de nos da lista
*/
void inserirNaListaSimples(lineList* list ,string lineName){
	lineListNode* listNode = new lineListNode; //O(1)
	listNode->lineName = lineName; //O(1)
	listNode->next = NULL; // O(1)

	if(list->firstNode == NULL){ //O(1)
	list->firstNode = listNode; // O(1)
	list->size = 1; // O(1)
	return; //O(1)
	}

	lineListNode* lastObject = list->firstNode; //O(1)
	while (lastObject->next != NULL){ // O(n)
	lastObject = lastObject->next; // O(1)
	}

	lastObject->next = listNode; //O(1)
	list->size = list->size + 1; //O(1)
}

//Essa funcao busca por uma Linha de Onibus especifica
// com base no indice dela, retornando um ponteiro para
// o no no indice especificado ou nullptr se estiver fora
// dos limites
/*
No pior caso, a Complexidade dessa função é O(n) devido ao Looping
while onde percorre o numero de nó da Lista.
As demais operações como comparações são constantes, sendo assim O(1)
*/
lineListNode* pegarDoIndexListaSimples(lineList* list, int index){
	lineListNode* obj = list->firstNode; // O(1)

	if(obj == NULL){ //O(1)
	return NULL; // O(1)
	}
```

```
	if(index == 0){ //O(1)
	return obj; // O(1)
	}
	int ct_index = 0; //O(1)

	while(ct_index != index && (list != NULL)){ // O(n)
	obj = obj->next; //O(1)
	ct_index++; //O(1)
	}

	return obj; //O(1)
}

//Essa funcao remove uma Linha de Onibus (no) especifico com
// base no seu indice, liberando memoria alocada para o no de Linha
// e atualiza os ponteiros
/*
No pior caso, a função tem de complexidade O(n) devido ao while que percorre
o numero de nós da Lista de Linhas. As demais funções são operações constantes
sendo assim O(1).
*/
void apagarNoIndexListaSimples(lineList* list, int index){
	lineListNode* obj = list->firstNode; //O(1)
	lineListNode* prev = obj; // O(1)

	if(obj->next == NULL){ //O(1)
	list->firstNode = NULL; // O(1)
	delete obj; // O(1)
	list->size=0; // O(1)
	return; // O(1)
	}

	if(index == 0){ // O(1)
	obj = obj->next; // O(1)

	list->firstNode = obj; // O(1)
	delete prev;// O(1)
	list->size--; // O(1)
	return; // O(1)
	}
```

```
        int ct_index = 0; // O(1)

        while(ct_index != index && (list != NULL)){ // O(n)
        prev = obj; // O(1)
        obj = obj->next; // O(1)
        ct_index++; // O(1)
        }

        prev->next = obj->next; // O(1)
        delete obj; // O(1)
        list->size--; // O(1)

}

//Imprime a lista de Linhas de Onibus
/*
A complexidade dessa função no pior caso é O(n) por percorrer a Lista de nós
(Linhas)
*/
void ImprimirListaSimples(lineList* list) {
        lineListNode* currentNode = list->firstNode; // O(1)

        while (currentNode != nullptr) { //O(n)
        std::cout << "Linha: " << currentNode->lineName << endl; // O(1)

        currentNode = currentNode->next; // O(1)
        }
}

//Funcao que cria uma lista circular de
// paradas para um Linha de onibus especifica
// aloca memoria para hourRoundList e inicialiaza
/*
A complexidade dessa função é O(1). Ela possui operações constantes
resultando em complexidade constante
*/
hourRoundList* criarListaCircular(){
        hourRoundList* list = new hourRoundList; // O(1)
        list->firstNode = NULL; // O(1)
        list->lastNode = NULL; // O(1)
        list->size = 0; // O(1)
        return list; // O(1)
```

```
}

//Funcao que adiciona uma nova parada a lista Circular depois de criada
// ela adiciona um novo no de parada e o insere no final da lista
/*
A complexidade dessa função é O(1) por manter operações constantes, por ser
um conjunto de O(1) então seu todo permanece O(1)
*/
void inserirNaListaCircular(hourRoundList* list, string nome, string saida, string
chegada){
	hourRoundNode* node = new hourRoundNode; // O(1)

	node->chegada = chegada; // O(1)
	node->nome = nome; // O(1)
	node->saida = saida; // O(1)

	if(list->firstNode == NULL){ // O(1)
	node->next = node; // O(1)
	list->firstNode = node; // O(1)
	list->lastNode = node; // O(1)
	node->prev = node; // O(1)
	}else{
	list->lastNode->next = node; // O(1)
	list->firstNode->prev = node; // O(1)
	node->prev = list->lastNode; // O(1)
	node->next = list->firstNode; // O(1)

	list->lastNode = node; // O(1)
	}
	list->size ++; // O(1)
}

//Funcao que remove uma parada especifica em uma Linha de Onibus
//Ela faz a busca com base no nome da parada e a remove da lista Circular
/*
A complexidade é dada pela busca de Linhas e de Paradas (O(n+m)).
primeiramente O(n) pela busca de Linhas a partir da função
pegardoIndexListaSimples
e para a busca de paradas O(n), assim O(m), por percorrer a Lista de paradas na
Linha
assim sendo : O(n+m)
*/
```

```
void removerParada(lineList* lista, int indiceLinha, const std::string& nomeParada){
        lineListNode* linha = pegarDoIndexListaSimples(lista, indiceLinha); // O(n)

        if(linha != nullptr){
        hourRoundList* paradas = linha->parada; // O(1)

        if(paradas != nullptr){
        hourRoundNode* paradaAtual = paradas->firstNode; // O(1)
        hourRoundNode* paradaAnterior = nullptr; // O(1)

        while (paradaAtual != nullptr) //O(n)
        {
        if(paradaAtual->nome == nomeParada){  // O(1)
                if(paradaAnterior == nullptr){ // O(1)
                paradas->firstNode = paradaAtual->next; // O(1)
                delete paradaAtual; // O(1)
                paradas->size--; // O(1)
                cout << "Parada removida da linha " << linha->lineName << endl; // O(1)
                return; // O(1)
                }
                else{
                paradaAnterior->next = paradaAtual->next; // O(1)
                delete paradaAtual; // O(1)
                paradas->size--; // O(1)
                cout << "Parada removida da linha " << linha->lineName << endl; // O(1)
                return; // O(1)
                }
        }
        paradaAnterior = paradaAtual; // O(1)
        paradaAtual = paradaAtual->next; // O(1)
        }
        //se chegou ate aqui eh porque a parada nao foi encontrada
        cout << "Parada nao encontrada na Linha " << linha->lineName << endl; // O(1)
        }
        //Caso em que a linha esteja vazia, sem paradas
        else{
        cout << "A Linha " << linha->lineName << " nao possui paradas" << endl; // O(1)
        }
        }
        //Situacao em que nao encontra a linha a qual deseja remover a parada
        else{
        cout << "Linha nao encontrada na posicao: " << indiceLinha << endl; // O(1)
```

```
        }
}

//Funcao que altera os dados de uma parada especifica em uma Linha de Onibus
// Ela faz a busca com base no nome da parada e apos isso atualiza seus dados
/*
Da mesma forma da função anterior, ela percorre primeiro as Linhas O(n) e depois
as
paradas O(m), sendo  as demais operações constantes O(1).
Portanto a Complexidade dela é O(n+m)
*/
void alterarParada(lineList* lista, int indiceLinha, const std::string& nomeParada,
const std::string& novoNome, const std::string& novaSaida, const std::string&
novaChegada) {
        lineListNode* linha = pegarDoIndexListaSimples(lista, indiceLinha); //O(n)

        if (linha != nullptr) {
        hourRoundList* paradas = linha->parada; // O(1)

        if (paradas != nullptr) {
        hourRoundNode* paradaAtual = paradas->firstNode; // O(1)

        while (paradaAtual != nullptr) {//O(n)
        //alteracao dos dados no looping
        if (paradaAtual->nome == nomeParada) { // O(1)
                paradaAtual->nome = novoNome; // O(1)
                paradaAtual->saida = novaSaida; // O(1)
                paradaAtual->chegada = novaChegada; // O(1)
                cout << "Dados da parada alterados com sucesso na Linha " <<
linha->lineName << endl; // O(1)
                return; // O(1)
        }
        paradaAtual = paradaAtual->next; // O(1)
        }
        // Se chegou até aqui, a parada não foi encontrada
        cout << "Parada não encontrada na Linha " << linha->lineName << endl; // O(1)
        }
        //caso em que a linha não possua paradas
        else {
        cout << "A Linha " << linha->lineName << " não possui paradas" << endl; // O(1)
        }
        }
```

```
        // Situação em que não encontra a linha a qual deseja alterar a parada
        else {
        cout << "Linha não encontrada na posição: " << indiceLinha << endl; // O(1)
        }
}

//Funcao que permite adicionar uma nova parada a uma Linha de Onibus especifica
// Criando um novo no de parada e adicionando a lista Circular de paradas
/*
A complexidade dessa função é O(n) por percorrer apenas a Lista de Linhas ao chamar a função pegarDoIndexListaSimples. As demais operações permanecem costantes
O(1)
*/
void atribuirParadaEmLinha(lineList* linhas, int indiceLinha, std::string nomeParada, std::string saidaParada, std::string chegadaParada){
        if(indiceLinha < 0 || indiceLinha >= linhas->size){ // O(1)
        std::cout << "Indice de linha invalido." << std::endl; // O(1)
        return; // O(1)
        }
        // Cria uma nova parada
        hourRoundNode* novaParada = new hourRoundNode(); // O(1)
        novaParada->nome = nomeParada; // O(1)
        novaParada->saida = saidaParada; // O(1)
        novaParada->chegada = chegadaParada; // O(1)
        novaParada->next = nullptr; // O(1)

        // Linha desejada
        lineListNode* linha = pegarDoIndexListaSimples(linhas, indiceLinha); //O(n)

        // Se a linha não possui uma lista de paradas, cria uma nova lista de paradas na linha
        if(linha->parada == nullptr){ // O(1)
        linha->parada = criarListaCircular(); // O(1)
        }
        // Adiciona a nova parada a lista de paradas da linha
        inserirNaListaCircular(linha->parada, nomeParada, saidaParada, chegadaParada); // O(1)
}

//Essa funcao faz a impressao da lista de paradas
/*
```

A complexidade dessa função no pior caso é O(n) pelo do...while ao percorrer a Lista de Paradas
*/
void ImprimirListaCircular(hourRoundList* list) {
	if (list->firstNode == nullptr) { // O(1)
	std::cout << "A lista de paradas está vazia." << std::endl; // O(1)
	return; // O(1)
	}

	hourRoundNode* currentNode = list->firstNode; // O(1)

	cout << "Horarios de Parada:" << endl; // O(1)
	do{
	cout << "Nome: " << currentNode->nome << ", Saida: " << currentNode->saida << ", Chegada: " << currentNode->chegada << endl; // O(1)
	currentNode = currentNode->next; // O(1)
	}while(currentNode->next != list->firstNode); //O(n)
}

//Ja nessa funcao, ela imprime todas as Linhas de Onibus com suas respectivas paradas
// percorredo a lista de Linhas de Onibus e a cada Linha, imprimindo suas paradas
/*
Por Imprimir as Linhas com suas Paradas, a complexidade desta função no pior caso é O(n*m), onde
n é a lista de Linhas pelo while e imprimir-lo e m é ao chamar a função de imprimirListaCircular
que como se trata tambem de impressão fica: O(n*m)
*/
void imprimirLinhasComParadas(lineList* linhas){
	if(linhas->size == 0){ // O(1)
	cout << endl << "A lista de linhas esta vazia" << endl; // O(1)
	return; // O(1)
	}
	lineListNode* linhaAtual = linhas->firstNode; // O(1)

	cout << endl << "Lista de Linhas de Onibus com suas respectivas paradas:" << endl; // O(1)

	while(linhaAtual != nullptr){ // O(n)
	cout << "Linha: " << linhaAtual->lineName << endl; // O(1)
	if(linhaAtual->parada != nullptr){ // O(1)

```
        ImprimirListaCircular(linhaAtual->parada);//O(n)
        }
        else{
        cout << "Esta linha nao possui paradas atribuidas" << endl; // O(1)
        }

        linhaAtual = linhaAtual->next;  // O(1)
        }
}

//Função da senha para Manutenção do programa
/*
Complexidade O(1) pela operação constante
*/
  bool checkPassword (const std::string & corretPassword, const std::string &
userPassword) //Funcao para verificar a senha em caso de manutencao de linhas e
paradas
  {
        return corretPassword == userPassword;//O(1)
  }
//faz a busca da função com seu numero da linha retornando
// a posição na qual a linha está
/*
A complexidade no pior caso dessa função é O(n), devido ao while onde
pode chegar até o ultimo nó.
*/
int BuscaLinha(lineList* linha,int &numeroLinha){
        lineListNode* list = linha->firstNode; //O(1)
        int aux = 0; //O(1)
        while (list != nullptr && list->numberLine != numeroLinha) // O(n)
        {
        list = list->next; //O(1)
        aux++; //O(1)
        }
        if (list == nullptr) // O(1)
        {
        return -1;//O(1)
        }
        else{
        return aux; // O(1)
        }
}
```

BREVE EXPLICAÇÃO DE COMO AS ESTRUTURAS DE DADOS SE RELACIONAM

Como dito anteriormente todas as classes se relacionam para que haja uma gestão eficiente das informações sobre as linhas de ônibus e suas respectivas parada. A hourRoundNode é usado para representar paradas individuais em uma linha de ônibus, sendo parte integrante de uma lista circular, o que está diretamenteligada à hourRoundList que gerencia a lista circular de paradas de ônibus para uma linha específica, composta por múltiplas instâncias de hourRoundNod. Vale ressaltar que lineListNode representa linhas individuais em uma lista simples, cada uma associada a uma lista de paradas em hourRoundList. Ou seja, a lineList organiza a lista de linhas de ônibus, mantendo referências a nós de linha (lineListNode) e compartilhando a lista de paradas em hourRoundList para todas as linhas.

DIVISÃO DA TAREFA ENTRE OS MEMBROS

Foi um trabalho totalmente feito em dupla, ou seja, nossos conhecimentos foram compartilhados e tanto o código quanto o relatório teve participação igualitária de ambas.

DIFICULDADES ENCONTRADAS

- Na criação do arquivo txt, visto que os vídeos que foram vistos para a sua criação só continham exemplos simples. Foram realizadas tentativas mas não houve sucesso na execução.
- No desenvolver das relações entre as classes.
- Ter que começar o projeto do zero no meio do prazo devido a um erro de interpretação no início
- Conciliar o prazo

CONCLUSÕES

O projeto visou desenvolver um sistema de informação para fornecer dados atualizados sobre linhas de ônibus a passageiros em terminais de ônibus. Através o uso das estruturas de dados tornou-se capaz de armazenar informações sobre linhas e paradas, com a respectiva capacidade de busca e manutenção. A alocação dinâmica de memória e a leitura de dados de arquivos de texto tornaram o funcionamento do sistema flexível e fácil de manter. Com base nas dificuldades que

surgiram no decorrer da criação do projeto, foi preciso aguçar os conhecimentos em listas e ponteiros, tal como a experiência de exercer o trabalho no prazo estabelecido.

REFERÊNCIAS BIBLIOGRÁFICAS

https://youtu.be/BqmGWsnuHP0?si=Hc6-TIJLw5ej04nE

https://youtu.be/AQfAOFywDeM

https://youtu.be/AJzP6N_8KxY?si=y5DEuTbDYjryV0fN

https://youtu.be/_LlxxU3vfpg?si=X0LdT1b_S14_RxQU

https://drive.google.com/file/d/1a5wubDrl8N-C_Koit-78MM01xACqYKxN/view